Achieving Product Qualities Through Software Architecture Practices

Presentation for

CSEE&T 2004

Mar 3, 2004

Linda Northrop
Director, Product Line Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This work is sponsored by the U.S. Department of Defense.

© 2004 by Carnegie Mellon University

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
03 MAR 2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Achieving Product Qualities Through Software Architecture Practices	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,Software Engineering Institute,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
17th Conference on Software Engineering Education and Training (CSEE&T 2004) Norfolk, VA, March 1-3, 2004.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**87**	

Carnegie Mellon
Software Engineering Institute

Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Software Engineering Institute

Applied R&D laboratory situated as a college-level unit at Carnegie Mellon University, Pittsburgh, PA, USA

Established in 1984

Technical staff of 335

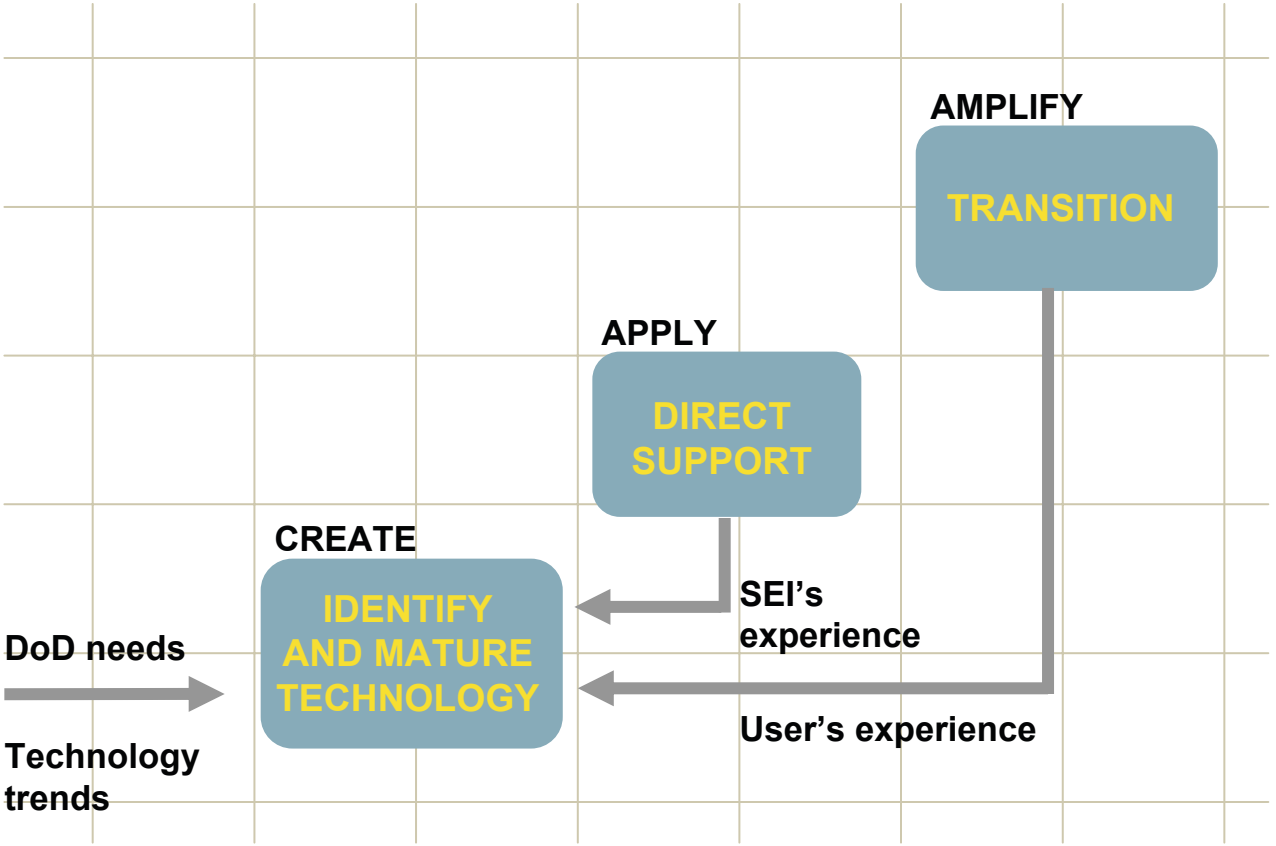
Offices in Pittsburgh, Pennsylvania (USA), Arlington, Virginia (USA) and Frankfurt Germany

Purpose: Help others improve their software engineering practices

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

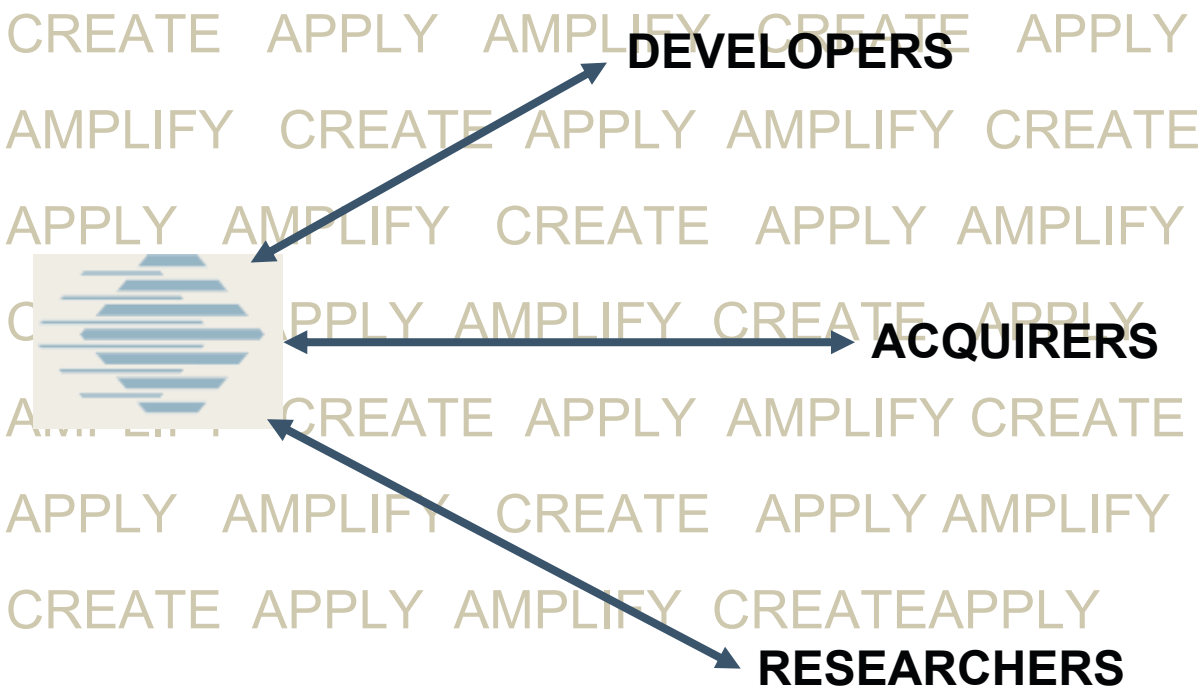
SEI's Strategic Functions



© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

SEI and the Community



© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Product Line Systems Program

Our Goal: To enable widespread product line practice through architecture-centric development

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Our Strategy

Software Architecture

(Software Architecture Technology Initiative)

Software Product Lines

(Product Line Practice Initiative)

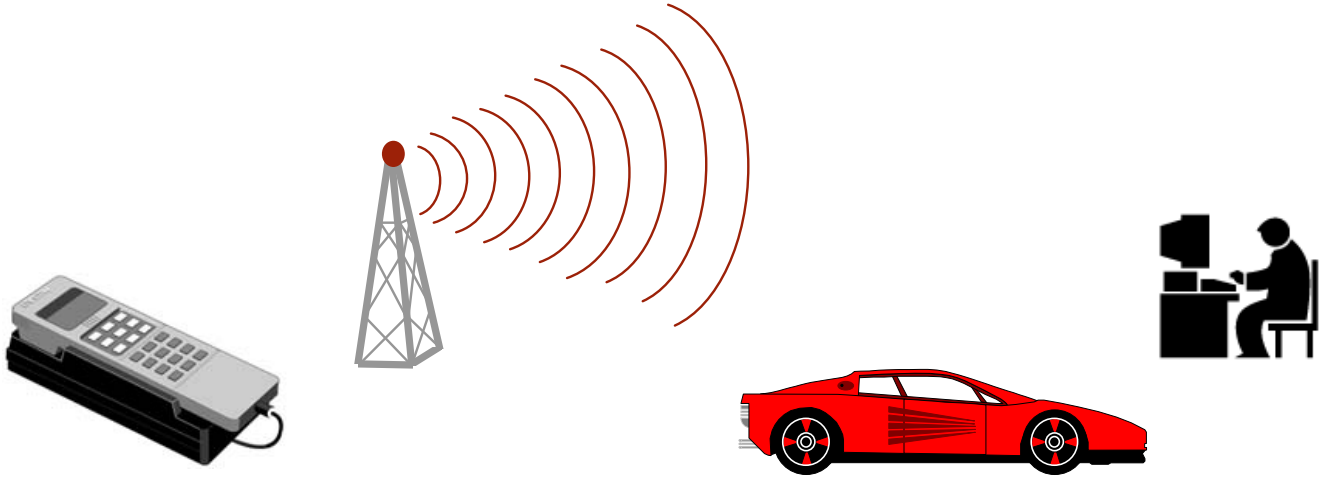
Component Technology

(Predictable Assembly from Certifiable Components Initiative)

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Business Success Requires Software Prowess



Software pervades every sector. Software has become the bottom line for many organizations who never envisioned themselves in the software business.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Business Goals

High quality

Quick time to market

Effective use of limited resources

Product alignment

Low cost production

Low cost maintenance

Mass customization

Mind share

improved efficiency and productivity

© 2004 by Carnegie Mellon University

The Ultimate Universal Goal

© 2004 by Carnegie Mellon University

Software Strategies Are Needed

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2004 by Carnegie Mellon University

Software Architecture: Common Ideas

A software architecture is a “first-cut” at designing the system and solving the problem or fitting the need.

A software architecture is an ad hoc box-and-line drawing of the system that is intended to solve the problems articulated by the specification.

- Boxes define the elements or “parts” of the system.
- Lines define the interactions or between the parts.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Our Definition of Software Architecture

"*The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.*"

Bass L.; Clements P.; Kazman R. *Software Architecture in Practice 2nd Edition* Reading, MA: Addison-Wesley, 2003.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Implications of Our Definition

Architecture is an abstraction of a system.

Systems can and do have many structures.

Every system *has* an architecture.

Just having an architecture is different from having an architecture that is known to everyone.

If you don't explicitly develop an architecture, you will get one anyway – *and you might not like what you get!*

© 2004 by Carnegie Mellon University

Why is Software Architecture Important?

Represents *earliest* design decisions

- hardest to change
- most critical to get right
- communication vehicle among stakeholders

First design artifact addressing

- performance
- modifiability
- reliability
- security

Key to systematic *reuse*

- transferable, reusable abstraction

The right architecture paves the way for system success.

The wrong architecture usually spells some form of disaster.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2004 by Carnegie Mellon University

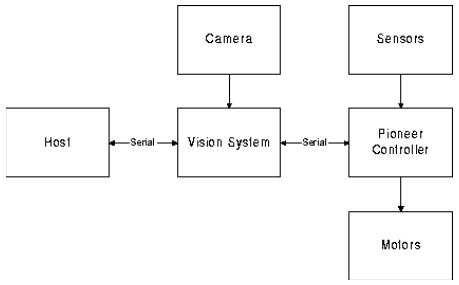
Carnegie Mellon
Software Engineering Institute

Requirements Beget Design

Requirements in various forms

Available knowledge

System

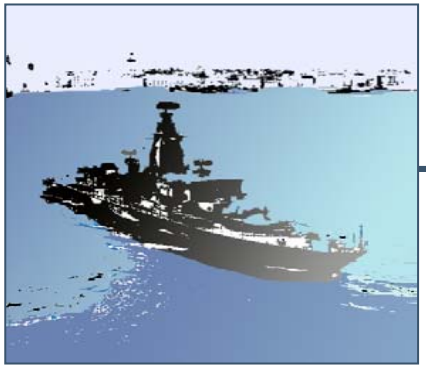


Designer

Architecture

© 2004 by Carnegie Mellon University

Software System Development



Functional Software Requirements

If function were all that mattered, any monolithic software would do, ..***but other things matter***…

The important quality attributes and their characterizations are key.

- Modifiability
- Interoperability
- Availability
- Security
- Predictability
- Portability
- :

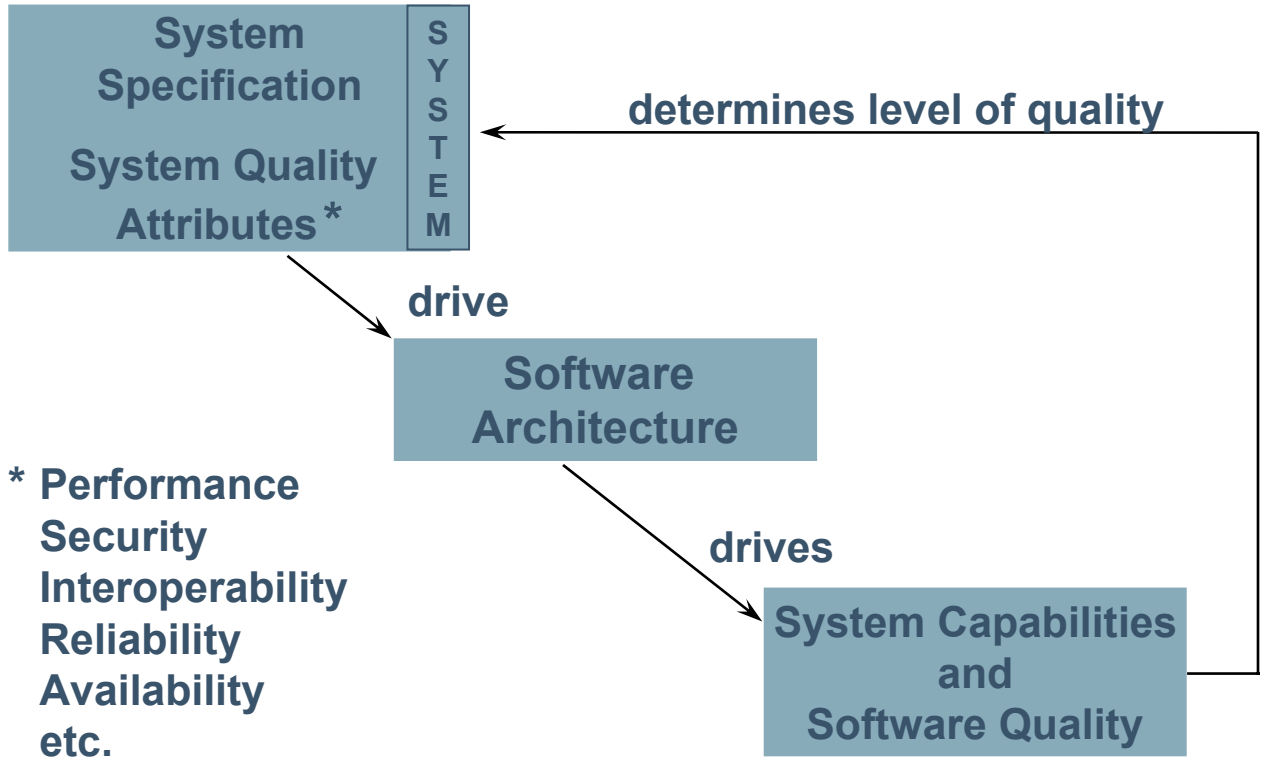
analysis, design, development

Quality Attribute Drivers → Software Architecture → Software

has these qualities

© 2004 by Carnegie Mellon University

System Qualities and Software Architecture



© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Architecture and Functionality

Functionality is largely orthogonal to quality attribute requirements.

- Functionality is the ability of a system to do the work it was intended to do.
- Systems are decomposed into elements to achieve a variety of purposes other than function.
  - Architectural choices promote certain qualities as well as implement the desired functionality.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Effects of Architectural Decisions on Quality Attributes

The degree to which a system meets it's quality attribute requirements is dependent on architectural decisions.

- A change in structure improving one quality often affects the other qualities.
- Architecture is critical to the realization of quality attributes.
- These product qualities should be designed into the architecture.
- Architecture can only permit, not guarantee, any quality attribute.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Challenges

What precisely do these quality attributes such as modifiability, security, performance, and reliability mean?

How do you architect to ensure the system will have its desired qualities?

Can a system be analyzed to determine these desired qualities?

How soon can such an analysis occur?

How do you know if software architecture for a system is suitable without having to build the system first?

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Quality Attribute Scenarios – 1

A solution to the problem of describing quality attributes is to use quality attribute scenarios as a means to better characterize quality attributes.

A quality attribute scenario consists of six parts.

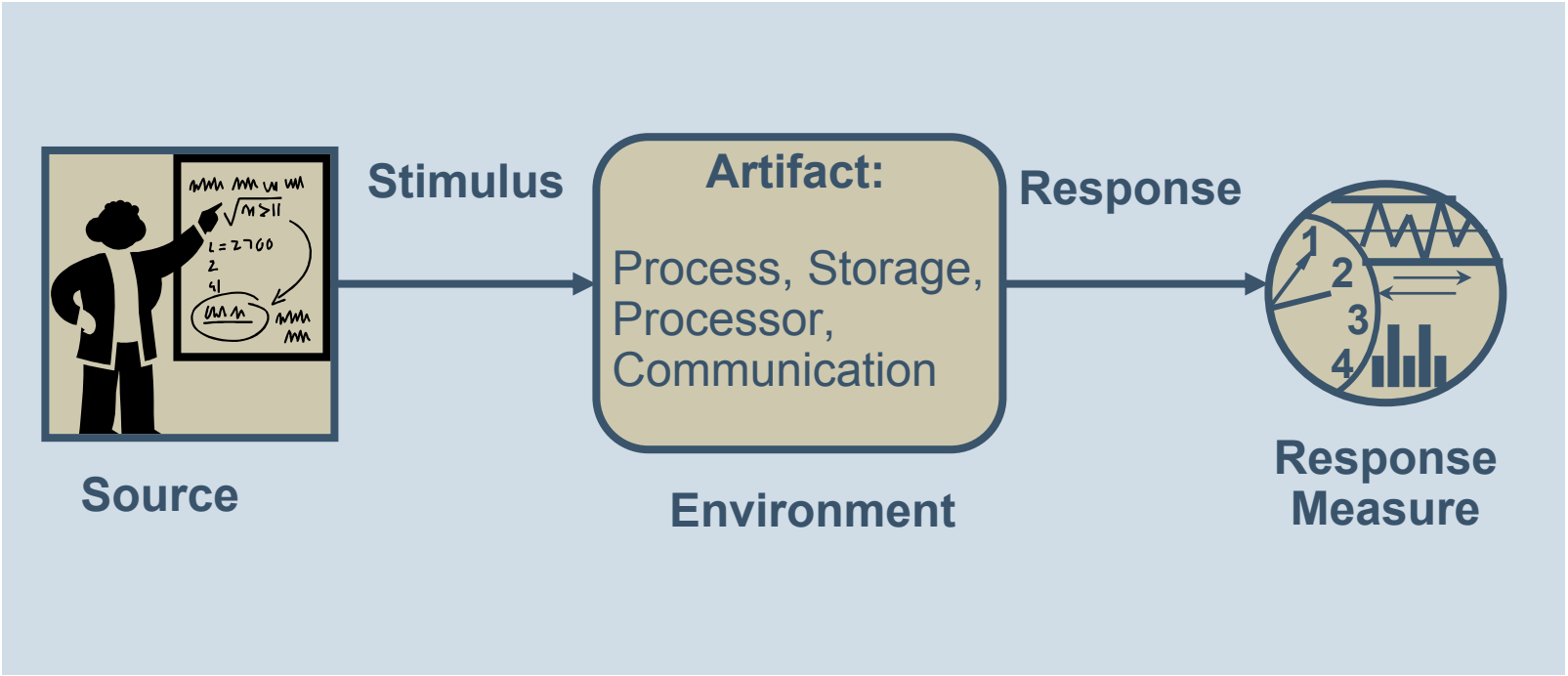
© 2004 by Carnegie Mellon University

Quality Attribute Scenarios – 2

1. stimulus – a condition that affects the system
2. response – the activity that results because of the stimulus
3. source of the stimulus – the entity that generated the stimulus
4. environment – the condition under which the stimulus occurred
5. artifact stimulated – the artifact that was stimulated by the stimulus
6. response measure – the measure by which the system's response will be evaluated

© 2004 by Carnegie Mellon University

Parts of a Quality Attribute Scenario



© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

General and Concrete Scenarios

General scenarios

- are those scenarios that are system independent
- represent quality attribute characterizations
- can be used to create concrete scenarios that are specific to a particular system.

General six-part scenarios exist for

- availability
- modifiability
- performance
- security
- testability
- usability

© 2004 by Carnegie Mellon University

Modifiability – 1

Definition: *Modifiability is about the cost of change and refers to the ease with which a software system can accommodate changes.*

Areas of concern include

- identifying what can change
  - functions, platforms, hardware, operating systems, middleware, systems it must operate with, protocols, and so forth
  - quality attributes: performance, reliability, future modifiability, and so forth
- When will the change be made and who will make it?

© 2004 by Carnegie Mellon University

Modifiability – 2

General scenario considerations:

Source	End user, developer, system administrator
Stimulus	Add/delete/modify functionality or quality attribute
Environment	Runtime, compile time, build time, design time
Artifacts	System: user interface, platform, environment, system that interoperates with target system

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Modifiability – 3

General scenario considerations (continued):

Response	• Locate places in the architecture to be modified. • Make modifications without affecting other functionality. • Test the modification with minimal effort. • Deploy the modification with minimal effort.
Response Measure	• Cost in terms of the number of affected components, effort, and money • Extent to which this modification affects other functions and/or quality attributes

© 2004 by Carnegie Mellon University

Sample Modifiability Scenario

A developer wishes to change the user interface (UI) code at design time. The modification is made with no side effects, in three hours.

Source	Developer
Stimulus	Wishes to change the UI
Artifact	Code
Environment	At design time
Response	Modification is made with no side effects
Response Measure	In three hours

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

The Reality About Software Architecture.

Quality attribute requirements are the primary drivers for architectural design.

The degree to which a system meets its quality attribute requirements is dependent on architectural decisions.

Software development needs to be driven by architectural decisions.

Architecture-centric development is key.

© 2004 by Carnegie Mellon University

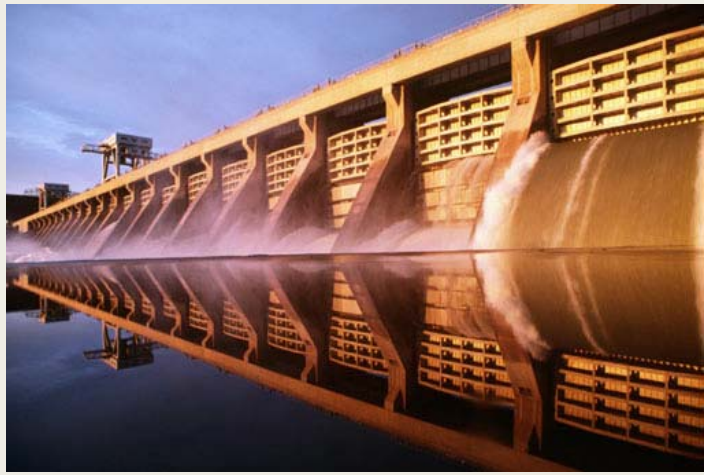
Carnegie Mellon
Software Engineering Institute

What is architecture-centric development?

Architecture-centric development involves

- Creating the business case for the system
- Understanding the requirements
- Creating or selecting the architecture
- Documenting and communicating the architecture
- Analyzing or evaluating the architecture
- Implementing the system based on the architecture
- Ensuring that the implementation conforms to the architecture
- Maintaining the architecture

The architecture must be both prescriptive and descriptive.



Influence of System Stakeholders - 1

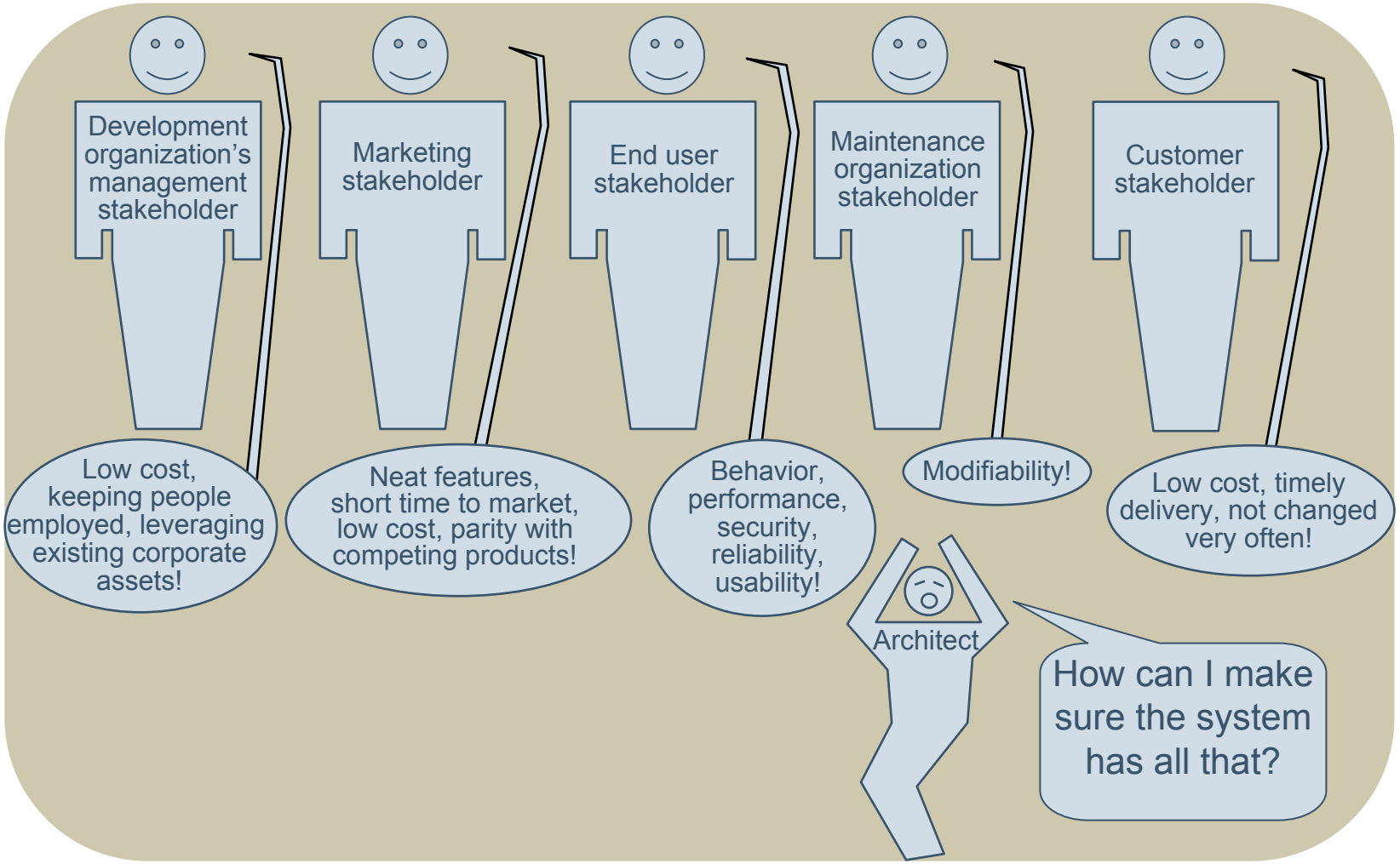
Stakeholders have an interest in the construction of a software system. Stakeholders might include

- customers
- users
- developers
- project managers
- marketers
- maintainers

Stakeholders have different concerns that they wish to guarantee and/or optimize.

© 2004 by Carnegie Mellon University

Influence of System Stakeholders – 2



© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Stakeholder Involvement

The organizational goals and the system properties required by the business are rarely understood, let alone fully articulated.

Customer quality attribute requirements are seldom documented, which results in

- goals not being achieved
- inevitable conflict between different stakeholders

Architects must identify and actively engage stakeholders in order to

- understand real constraints of the system
- manage the stakeholders' expectations
- negotiate the system's priorities
- make tradeoffs

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2004 by Carnegie Mellon University

SEI Work in Software Architecture: Maturing Sound Architecture Practices

Starting Points

Quality attribute/ performance engineering

Software Architecture Analysis Method (SAAM)

Security analysis

Reliability analysis

Software Architecture Evaluation Best Practices Report

Software architecture evaluations

Create

Technology

Attribute-specific patterns
Architecture expert

Life Cycle Practices

- Architectural requirements elicitation
- Architecture definition
- Architecture representation
- Architecture evaluation
- Architecture reconstruction

© 2004 by Carnegie Mellon University

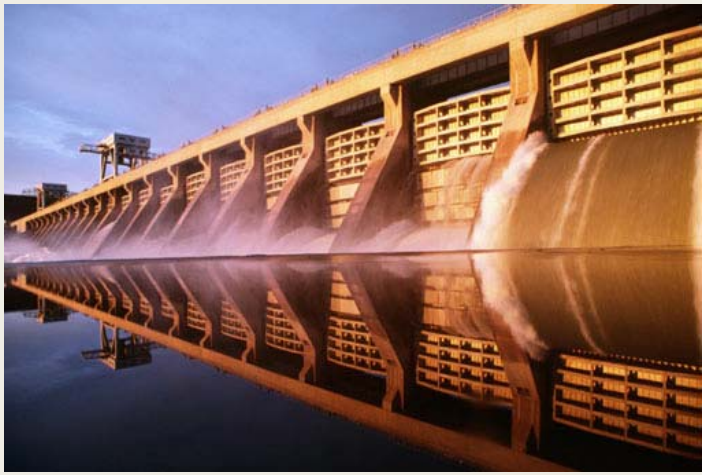
Carnegie Mellon
Software Engineering Institute

What is architecture-centric development?

Architecture-centric development involves

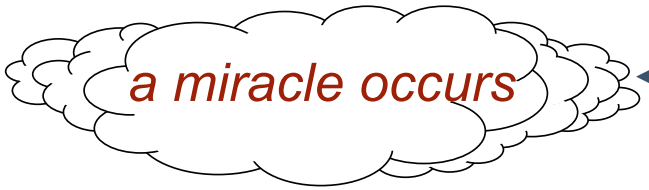
- Creating the business case for the system
- Understanding the requirements
- Creating or selecting the architecture
- Documenting and communicating the architecture
- Analyzing or evaluating the architecture
- Implementing the system based on the architecture
- Ensuring that the implementation conforms to the architecture
- Maintaining the architecture

The architecture must be both prescriptive and descriptive.



Traditional System Development

Operational descriptions
High level functional requirements
Legacy systems
New systems



Quality attributes are rarely captured in requirements specifications.

- *often vaguely understood*
- *often weakly articulated*

Specific system architecture
Software architecture

Detailed design
Implementation

© 2004 by Carnegie Mellon University

Quality Attribute Workshop

The Quality Attribute Workshop (QAW) is a facilitated method that engages system stakeholders early in the lifecycle to discover the driving quality attributes of a software intensive system.

Key points about the QAW are that it is

- system centric
- scenario based
- stakeholder focused
- used before the software architecture has been created

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

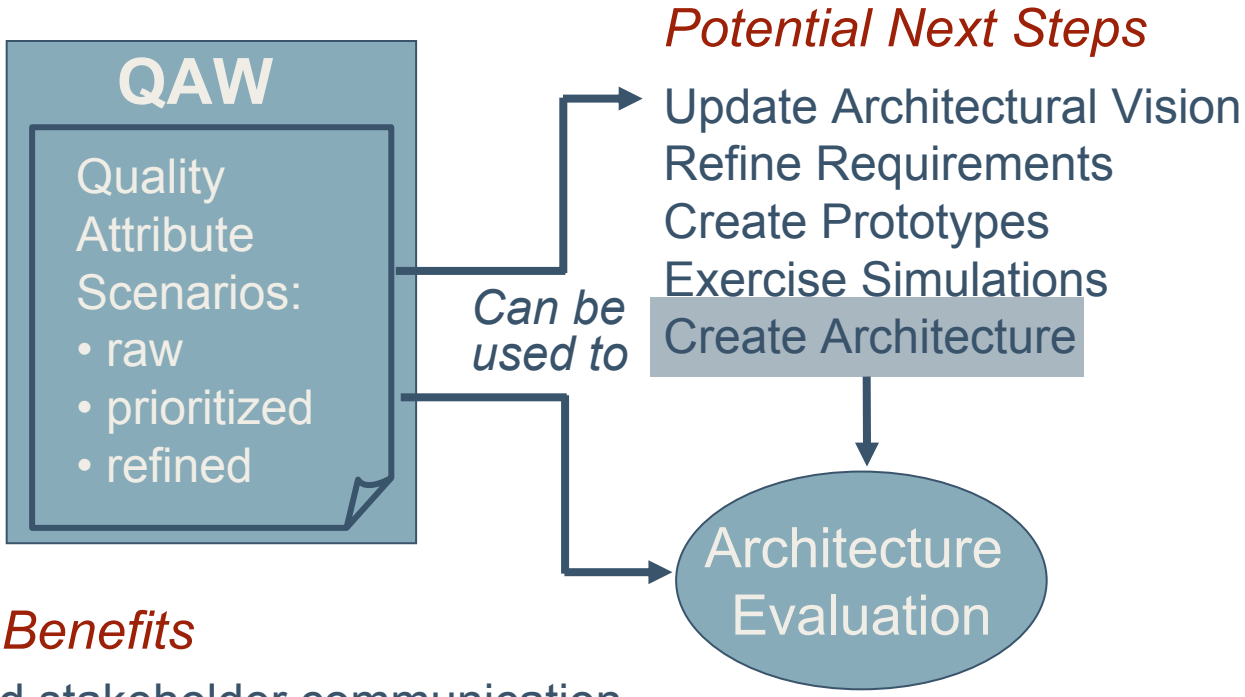
Quality Attribute Workshop Steps

1. Introductions and QAW Presentation
2. Business/Mission Presentation
3. Architecture Plan Presentation
4. Identify Architectural Drivers
5. Scenario Brainstorming
6. Scenario Consolidation
7. Scenario Prioritization
8. Scenario Refinement

Iterate as necessary with broader stakeholder community

© 2004 by Carnegie Mellon University

QAW Benefits and Next Steps



Potential Benefits

- Increased stakeholder communication
- Clarified quality attribute requirements
- Informed basis for architectural decisions

© 2004 by Carnegie Mellon University

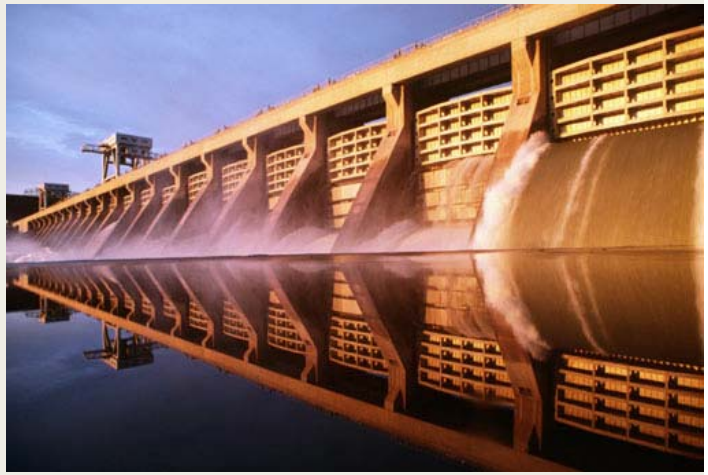
Carnegie Mellon
Software Engineering Institute

What Is Architecture-centric Development?

Architecture-centric development involves

- Creating the business case for the system
- Understanding the requirements
- Creating or selecting the architecture
- Documenting and communicating the architecture
- Analyzing or evaluating the architecture
- Implementing the system based on the architecture
- Ensuring that the implementation conforms to the architecture
- Maintaining the architecture

The architecture must be both prescriptive and descriptive.



Creating the Software Architecture

There are architecture definition methods and guidelines, many of which focus exclusively on the functional requirements.

It is possible to create an architecture based on the quality architectural drivers.

One way to approach this is to use architectural tactics and patterns and a method that capitalizes on both.

© 2004 by Carnegie Mellon University

Tactics

The design for a system consists of a collection of design decisions.

- Some decisions are intended to ensure the achievement of the functionality of the system.
- Other decisions are intended to help control the quality attribute responses.

These decisions are called *tactics*.

- A tactic is a design decision that is influential in the control of a quality attribute response.
- A collection of tactics is an *architectural strategy.*

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Tactics Catalog

Tactics have been defined for the following quality attributes:

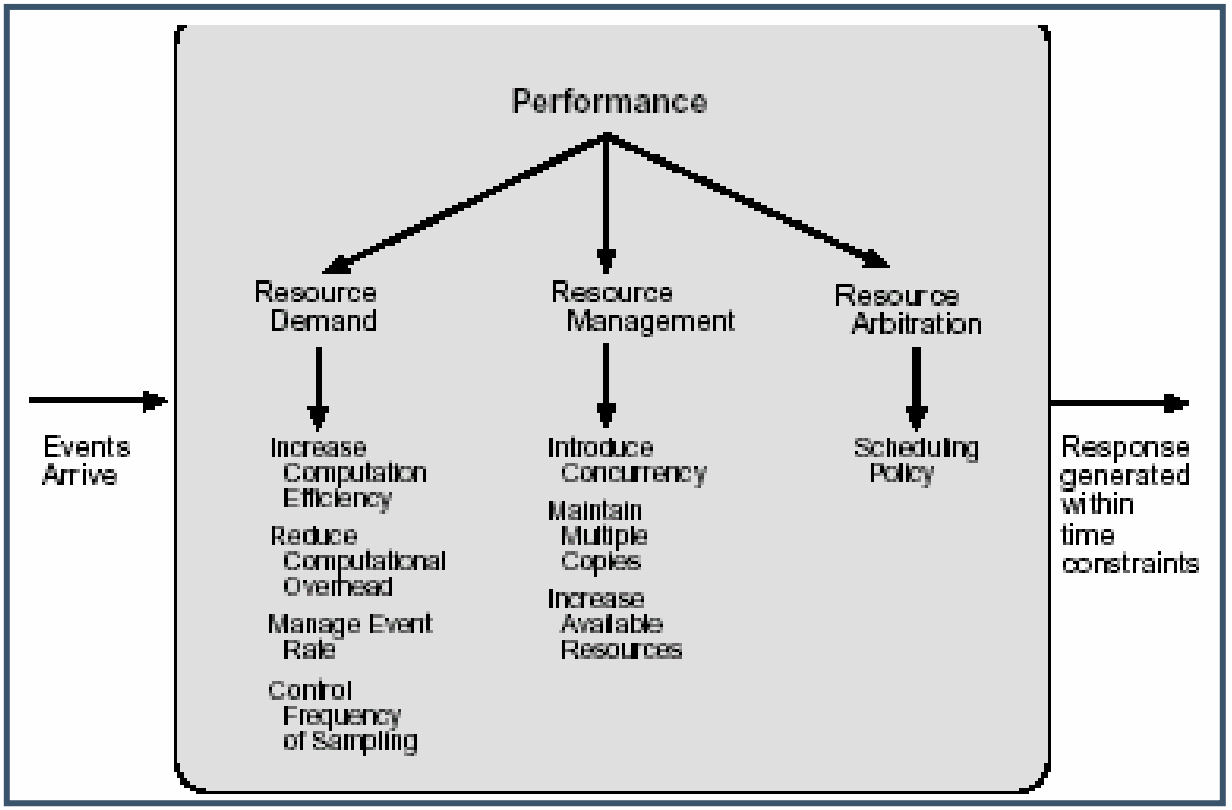
- Performance
- Availability
- Maintainability
- Usability
- Testability
- Security

Others are in the works.

© 2004 by Carnegie Mellon University

Performance Tactics

Summary of performance tactics



© 2004 by Carnegie Mellon University

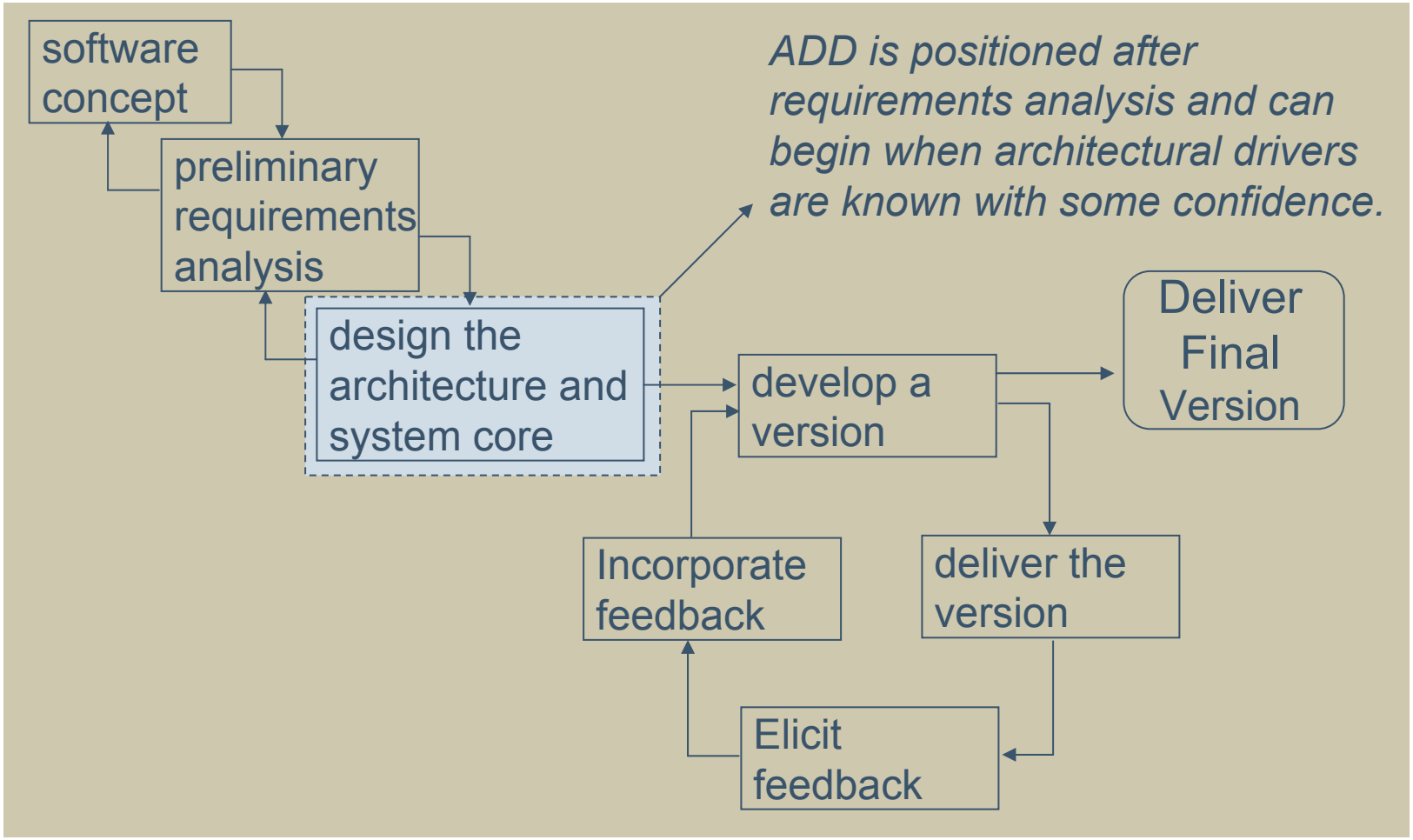
Attribute-Driven Design

The Attribute-Driven Design (ADD) method, developed at the SEI, is an approach to defining a software architecture that bases the decomposition process on the quality attributes the software must fill.

It follows a recursive decomposition process where, at each stage in the decomposition, tactics and architectural patterns are chosen to satisfy a set of quality scenarios.

© 2004 by Carnegie Mellon University

Evolutionary Delivery Life Cycle



© 2004 by Carnegie Mellon University

ADD Method's Inputs and Outputs

Inputs

- constraints
- functional requirements
- quality attribute requirements

Outputs

- first several levels of module decomposition
- various other views of the system as appropriate
- set of elements for functionality and the interactions among them

© 2004 by Carnegie Mellon University

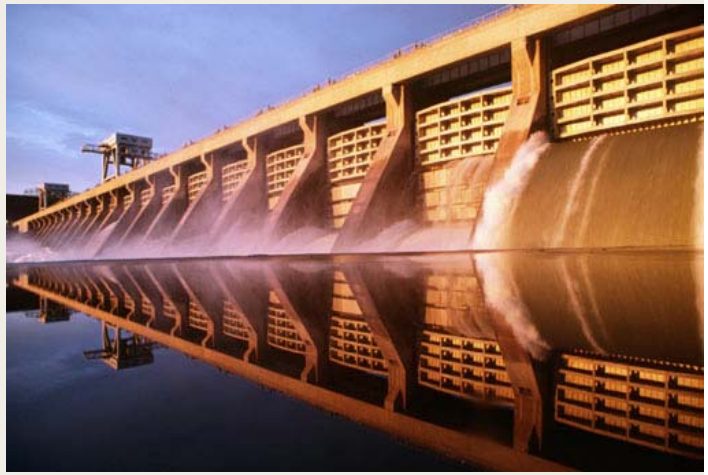
Carnegie Mellon
Software Engineering Institute

What Is Architecture-centric Development?

Architecture-centric development involves

- Creating the business case for the system
- Understanding the requirements
- Creating or selecting the architecture
- Documenting and communicating the architecture
- Analyzing or evaluating the architecture
- Implementing the system based on the architecture
- Ensuring that the implementation conforms to the architecture
- Maintaining the architecture

The architecture must be both prescriptive and descriptive.



Carnegie Mellon
Software Engineering Institute

Importance of Architecture Documentation

Architecture documentation is important if and only if *communication* of the architecture is important.

- How can an architecture be used if it cannot be understood?
- How can it be understood if it cannot be communicated?

Documenting the architecture is the crowning step to creating it.

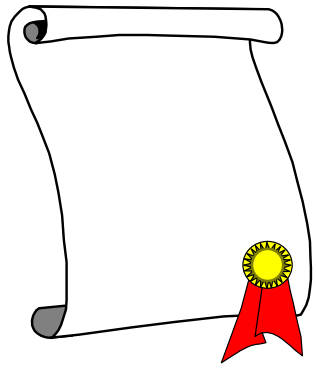
Documentation speaks for the architect, today and 20 years from today.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Seven Principles of Sound Documentation

Certain principles apply to all documentation, not just documentation for software architectures.



1. Write from the point of view of the reader.
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record rationale.
6. Keep documentation current but not too current.
7. Review documentation for fitness of purpose.

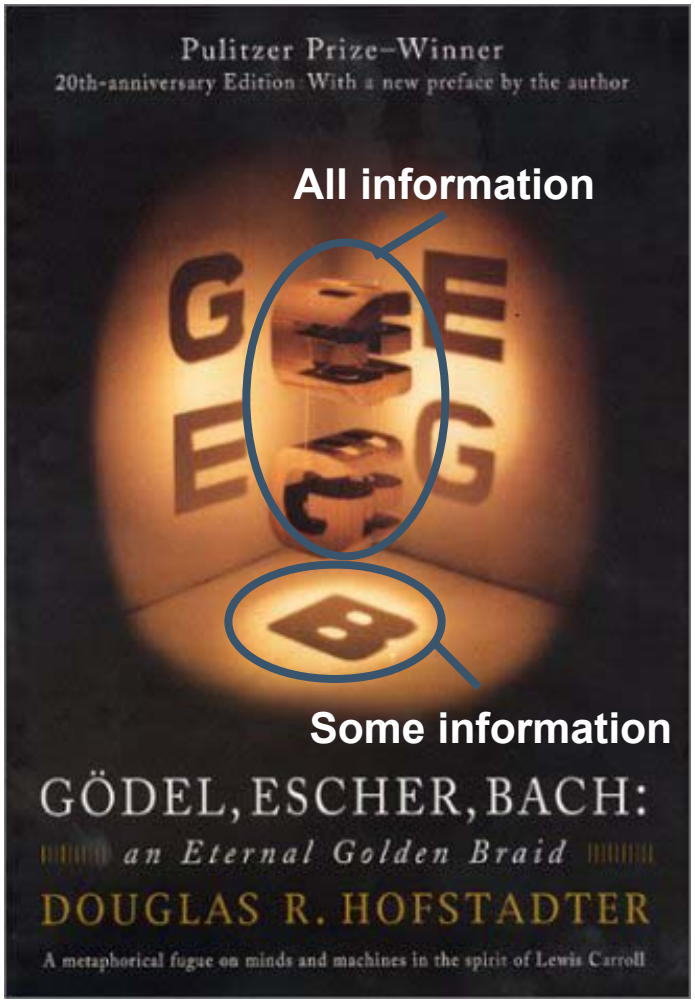
© 2004 by Carnegie Mellon University

Views

A view is a representation of a set of system elements and the relations associated with them.

Not *all* system elements, *some* of them.

A view binds an *element type* and *relation type* of interest, and illustrates them.

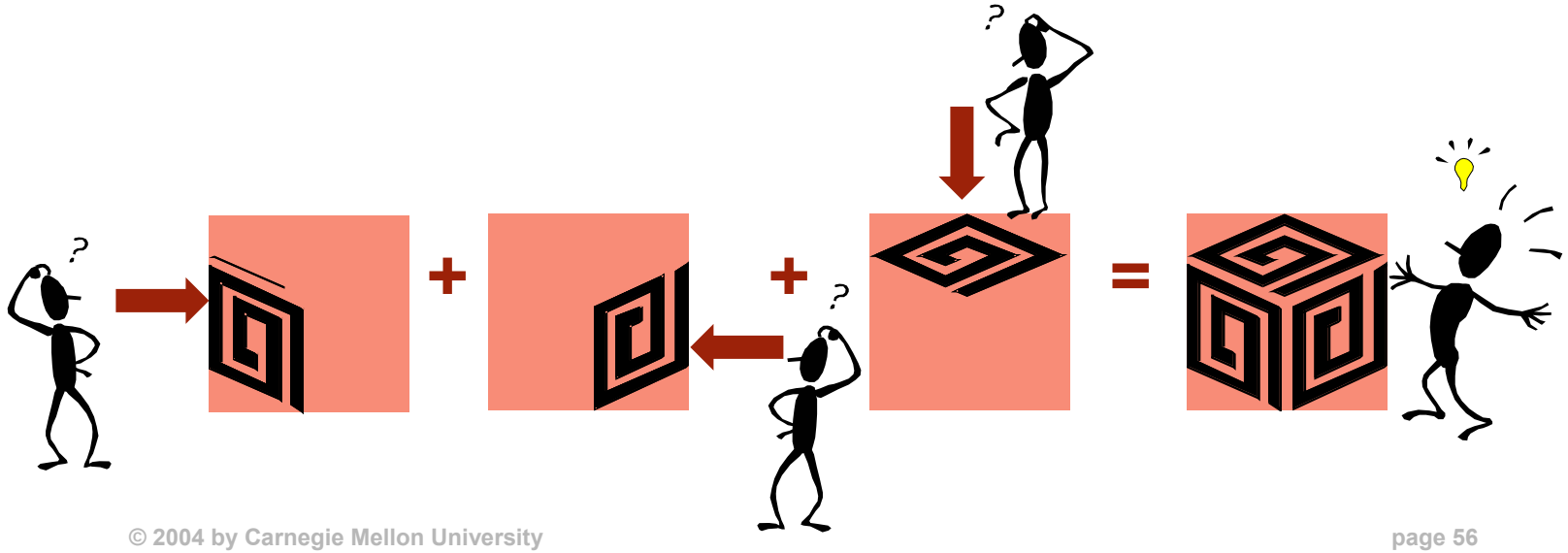


© 2004 by Carnegie Mellon University

View-Based Documentation

Views give us our basic principle of architecture documentation:

Documenting a software architecture is a matter of documenting the relevant views, and then adding information that applies to more than one view.



© 2004 by Carnegie Mellon University

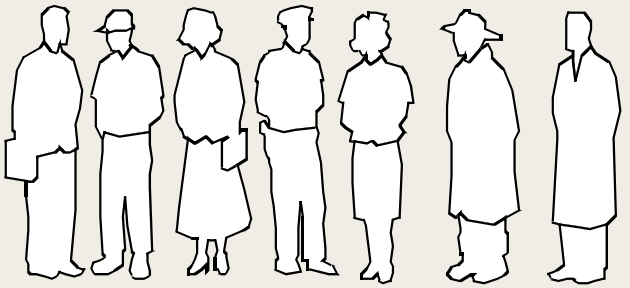
Which Views Are Relevant?

Which views are relevant? It depends on

- who the stakeholders are
- how they will use the documentation

Three primary uses for architecture documentation are

1. education - introducing people to the project
2. communication - among stakeholders
3. analysis - assuring quality attributes



© 2004 by Carnegie Mellon University

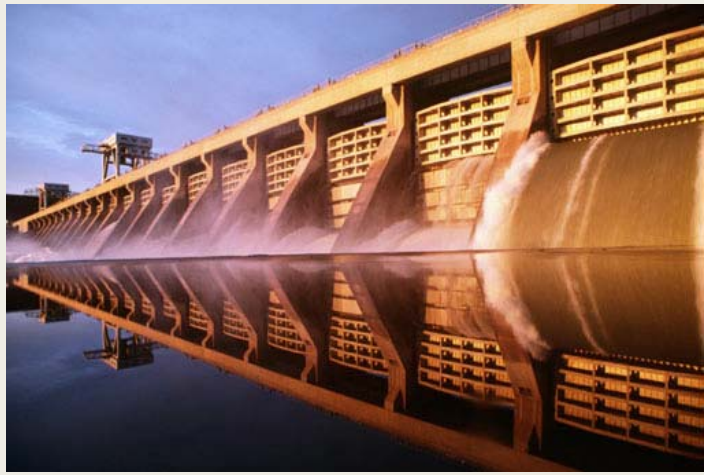
Carnegie Mellon
Software Engineering Institute

What Is Architecture-centric Development?

Architecture-centric development involves

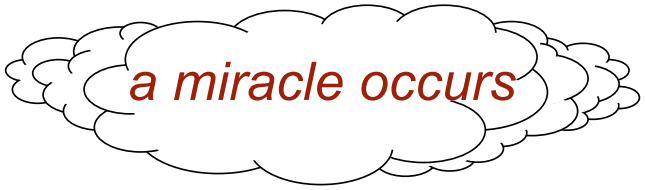
- Creating the business case for the system
- Understanding the requirements
- Creating or selecting the architecture
- Documenting and communicating the architecture
- Analyzing or evaluating the architecture
- Implementing the system based on the architecture
- Ensuring that the implementation conforms to the architecture
- Maintaining the architecture

The architecture must be both prescriptive and descriptive.



Traditional System Development

Operational descriptions
High level functional requirements
Legacy systems
New systems



Specific system architecture
Software architecture

Detailed design
Implementation

A Critical leap!

How do you know if the architecture is fit for purpose?

© 2004 by Carnegie Mellon University

Why Evaluate Architectures?

All design involves tradeoffs.

A software architecture is the earliest life-cycle artifact that embodies significant design decisions and tradeoffs.

- The earlier that risks are identified, the earlier that mitigation strategies can be developed potentially avoid the risks altogether.
- The earlier that defects are found, the less it costs to remove them.

© 2004 by Carnegie Mellon University

SEI's Architecture Tradeoff Analysis Method℠ (ATAM)℠

ATAM is an architecture evaluation method that

- focuses on multiple quality attributes
- illuminates points in the architecture where quality attribute *tradeoffs* occur
- generates a context for ongoing quantitative analysis
- utilizes an architecture's vested stakeholders as authorities on the quality attribute goals

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

The ATAM[SM]

The SEI has developed the Architecture Tradeoff Analysis Method[SM] (ATAM[SM]).

The purpose of ATAM is: *to assess the consequences of architectural decisions in light of quality attribute requirements and business goals*.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Purpose of ATAM – 1

The ATAM is a method that helps stakeholders ask the right questions to discover potentially problematic architectural decisions

Discovered risks can then be made the focus of mitigation activities: e.g. further design, further analysis, prototyping.

Surfaced tradeoffs can be explicitly identified and documented.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Purpose of ATAM – 2

The purpose of the ATAM is NOT to provide precise analyses . . . the purpose IS to discover risks created by architectural decisions.

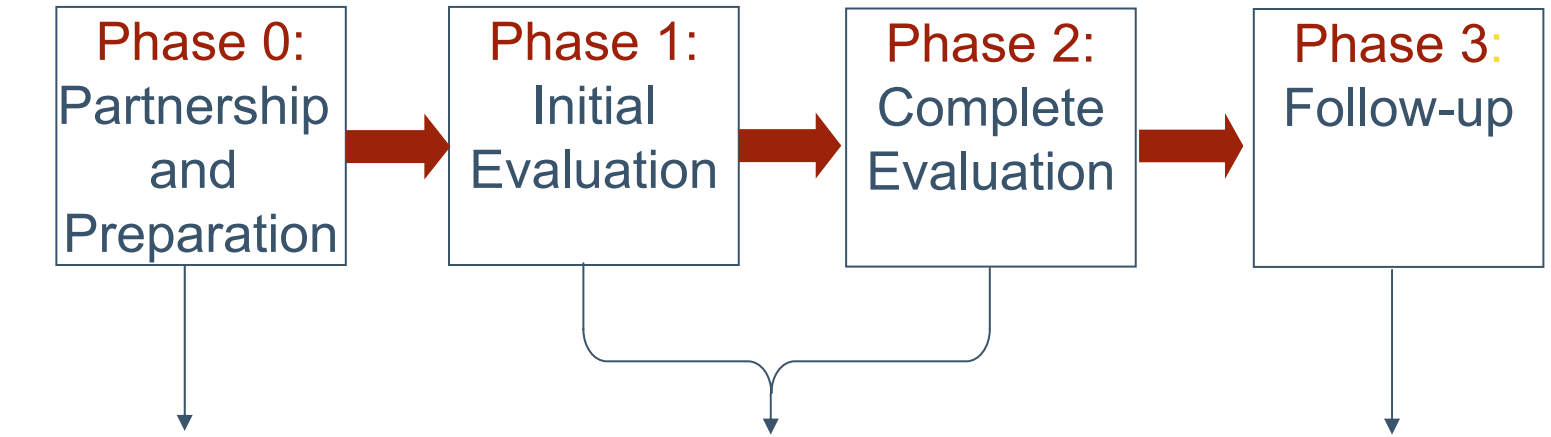
We want to find *trends*: correlation between architectural decisions and predictions of system properties.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

ATAM Phases

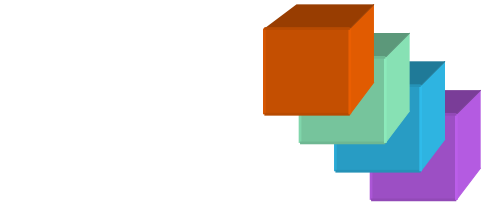
ATAM evaluations are conducted in four phases.



© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

ATAM Steps



1. Present the ATAM
2. Present business drivers
3. Present architecture

4. Identify architectural approaches
5. Generate quality attribute utility tree
6. Analyze architectural approaches

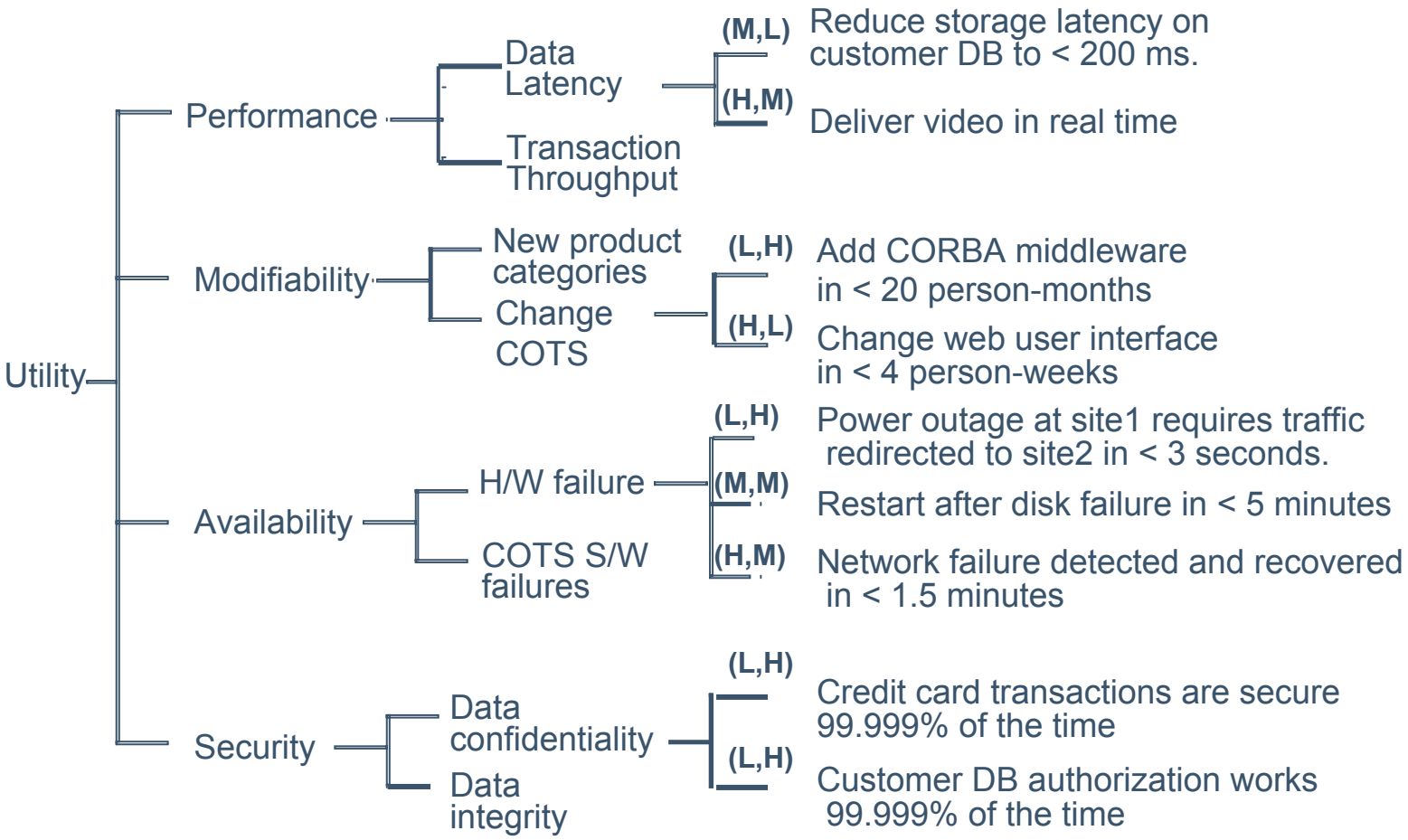
7. Brainstorm and prioritize scenarios
8. Analyze architectural approaches

9. Present results

© 2004 by Carnegie Mellon University

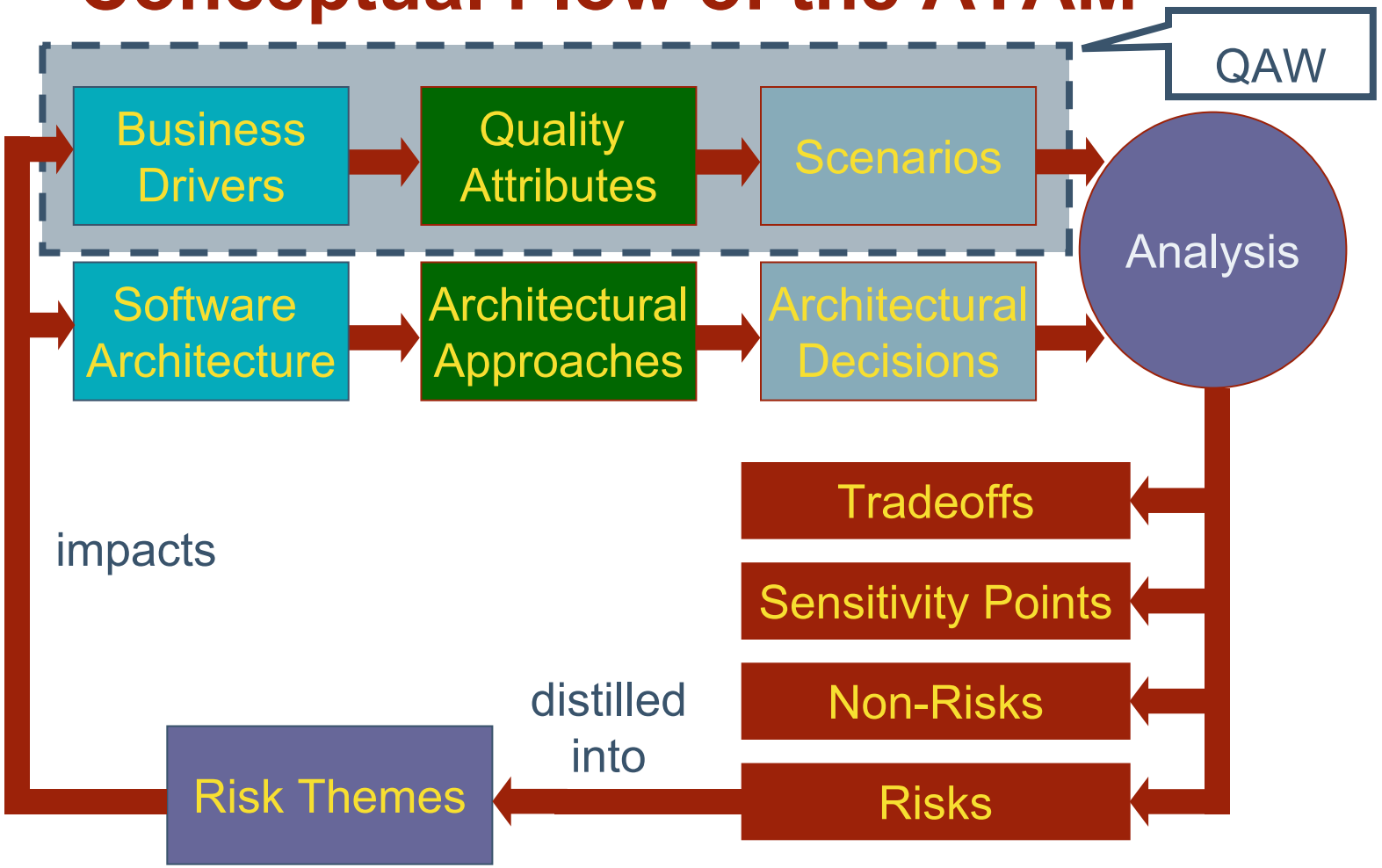
Carnegie Mellon
Software Engineering Institute

Example Utility Tree



© 2004 by Carnegie Mellon University

Conceptual Flow of the ATAM℠



© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

When to Use the ATAM

The ATAM can be used throughout the life cycle *when there is a software architecture to evaluate.*

The ATAM can be used

- after an architecture has been specified but there is little or no code
- to evaluate architectural alternatives
- to evaluate the architecture of an existing system

To perform an ATAM evaluation, *there must be a software architecture to evaluate*.

- An ATAM evaluation is inappropriate if the software architecture of the system has not been created yet.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

ATAM Benefits

The benefits of performing ATAM evaluations include

- clarified quality attribute requirements
- improved architecture documentation
- documented basis for architectural decisions
- identification of risks early in the life cycle
- increased communication among stakeholders

The result is improved architectures.

© 2004 by Carnegie Mellon University

Architecture Evaluation Experience

Benefits of early architecture evaluations

- Evaluations using the Architecture Tradeoff Analysis Method[SM] (ATAM[SM]) uncover an average 20 risks per two-day evaluation. Experience over a wide range of domains attributes these risks to
  - unknowns (requirements, hardware, COTS)
  - side effects of architectural decisions
  - improper architectural decisions
  - interactions with other organizations that provide system components
- Evaluations performed by AT&T have resulted in 10% productivity increase per project

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

SEI Work in Software Architecture: Maturing Sound Architecture Practices

Starting Points

Quality attribute/ performance engineering

Software Architecture Analysis Method (SAAM)

Security analysis

Reliability analysis

Software Architecture Evaluation Best Practices Report

Software architecture evaluations

Create

Technology

Attribute-specific patterns
Architecture expert

Life Cycle Practices

- Architectural requirements elicitation
- Architecture definition
- Architecture representation
- Architecture evaluation
- Architecture reconstruction

Apply/Amplify

Architecture Evaluations
Architecture Coaching
Architecture Reconstructions
Books
Courses
Certificate Programs
Acquisition Guidelines
Technical Reports
Web site
Workshops

© 2004 by Carnegie Mellon University

SEI Software Architecture Curriculum

Six courses

- Software Architecture: Principles and Practices
- Documenting Software Architectures
- Software Architecture Design and Analysis
- Software Product Lines
- ATAM Evaluator Training
- ATAM Facilitator Training

Three certificate programs

- Software Architecture Professional
- ATAM Evaluator
- ATAM Lead Evaluator

Coming in 2005: SEI Software Product Line Curriculum

© 2004 by Carnegie Mellon University

About the Curriculum

Software professionals can take individual courses based on specific needs or interests or complete one or more of the following three specially designed certificate programs:

- Software Architecture Professional
- ATAM[SM] Evaluator
- ATAM[SM] Lead Evaluator

The ATAM certificate programs qualify individuals to perform or lead SEI-authorized ATAM evaluations.

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Certificate Program Course Matrix

ATAM Lead Evaluator: 5 Courses & Coaching

Software Architecture Professional: 4 Courses	***Software Architecture: Principles and Practices***	***Documenting Software Architectures***	***Software Architecture Design and Analysis***	***Software Product Lines***
	ATAM Evaluator Training	***ATAM Facilitator Training***	***ATAM Coaching***	
	ATAM Evaluator 2 courses			

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

About all the Courses

All of the courses are two-day learning experiences that involve lectures and exercises.

The materials provided include books and class lecture slides.

Prerequisites are enforced.

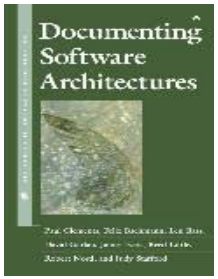
Any of the courses can also be scheduled for on site delivery.

© 2004 by Carnegie Mellon University

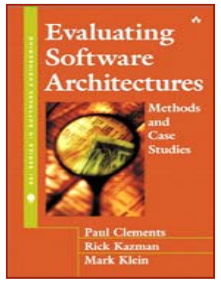
Associated Texts

Software Architecture in Practice, 2nd Edition

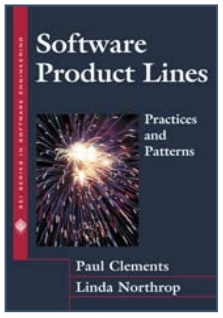
Documenting Software Architectures: Views and Beyond



Evaluating Software Architectures: Methods and Case Studies



Software Product Lines: Practices and Patterns



© 2004 by Carnegie Mellon University

SEI Software Architecture Workshop for Educators

August 16-18, 2004
Pittsburgh, PA

The Software Architecture Workshop for Educators is a three-day forum for sharing SEI software architecture technology with educators and for jointly determining ways to incorporate these concepts and methods into academic courses.

Schedule: Aug 16-17: Software Architecture: Principles and Practices Course
Aug 18: Facilitated Discussion

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Presentation Outline

Background

Software Architecture

Software Architecture Practices

Related Innovative Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Architecture Principles

Software architecture is important because it

- provides a communication vehicle among stakeholders
- is the result of the earliest design decisions
- is a transferable, reusable abstraction of a system

The degree to which a system meets its quality attribute requirements is dependent on architectural decisions.

Every software-intensive system *has* a software architecture. Just having an architecture is different from having an architecture that is known to everyone, much less one that is fit for the system's intended purpose.

An architecture-centric approach is critical to achieving and implementing an appropriate architecture.

© 2004 by Carnegie Mellon University

SEI Unique Contribution

The SEI work in software architecture technology and its associated methods are **notably unique** in their

- explicit focus on quality attributes
- direct linkage to business and mission goals
- explicit involvement of system stakeholders
- high-quality published materials for practitioner consumption
- grounding in state-of-the-art quality attribute models and reasoning frameworks

© 2004 by Carnegie Mellon University

The Total Picture

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

The Total Picture

© 2004 by Carnegie Mellon University

Conclusion

Software architecture is critical to achieving key product qualities.

Software architecture, product line practices, and predictable component practices hold great potential for achieving business and mission goals in the development of software-intensive systems.

© 2004 by Carnegie Mellon University

Contact Information

Linda Northrop
Director
Product Line Systems Program
Telephone: 412-268-7638
Email: lmn@sei.cmu.edu

U.S. mail:
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

World Wide Web:
http://www.sei.cmu.edu/ata
http://www.sei.cmu.edu/plp

SEI Fax: 412-268-5758

© 2004 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2004 by Carnegie Mellon University